THE LINUX PROGRAMMING INTERFACE

A Linux and UNIX® System Programming Handbook

MICHAEL KERRISK

no starch press

52

POSIX MESSAGE QUEUES

This chapter describes POSIX message queues, which allow processes to exchange data in the form of messages. POSIX message queues are similar to their System V counterparts, in that data is exchanged in units of whole messages. However, there are also some notable differences:

- POSIX message queues are reference counted. A queue that is marked for deletion is removed only after it is closed by all processes that are currently using it.
- Each System V message has an integer type, and messages can be selected in a variety of ways using *msgrcv()*. By contrast, POSIX messages have an associated priority, and messages are always strictly queued (and thus received) in priority order.
- POSIX message queues provide a feature that allows a process to be asynchronously notified when a message is available on a queue.

POSIX message queues are a relatively recent addition to Linux. The required implementation support was added in kernel 2.6.6 (in addition, *glibc* 2.3.4 or later is required).

> POSIX message queue support is an optional kernel component that is configured via the `CONFIG_POSIX_MQUEUE` option.

52.1 Overview

The main functions in the POSIX message queue API are the following:

- The *mq_open()* function creates a new message queue or opens an existing queue, returning a message queue descriptor for use in later calls.
- The *mq_send()* function writes a message to a queue.
- The *mq_receive()* function reads a message from a queue.
- The *mq_close()* function closes a message queue that the process previously opened.
- The *mq_unlink()* function removes a message queue name and marks the queue for deletion when all processes have closed it.

The above functions all serve fairly obvious purposes. In addition, a couple of features are peculiar to the POSIX message queue API:

- Each message queue has an associated set of attributes. Some of these attributes can be set when the queue is created or opened using *mq_open()*. Two functions are provided to retrieve and change queue attributes: *mq_getattr()* and *mq_setattr()*.
- The *mq_notify()* function allows a process to register for message notification from a queue. After registering, the process is notified of the availability of a message by delivery of a signal or by the invocation of a function in a separate thread.

52.2 Opening, Closing, and Unlinking a Message Queue

In this section, we look at the functions used to open, close, and remove message queues.

Opening a message queue

The *mq_open()* function creates a new message queue or opens an existing queue.

```
#include <fcntl.h>           /* Defines O_* constants */
#include <sys/stat.h>        /* Defines mode constants */
#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...
              /* mode_t mode, struct mq_attr *attr */);
```

Returns a message queue descriptor on success, or *(mqd_t)* –1 on error

The *name* argument identifies the message queue, and is specified according to the rules given in Section 51.1.

The *oflag* argument is a bit mask that controls various aspects of the operation of *mq_open()*. The values that can be included in this mask are summarized in Table 52-1.

Table 52-1: Bit values for the *mq_open() oflag* argument

Flag	Description
`O_CREAT`	Create queue if it doesn't already exist
`O_EXCL`	With `O_CREAT`, create queue exclusively
`O_RDONLY`	Open for reading only
`O_WRONLY`	Open for writing only
`O_RDWR`	Open for reading and writing
`O_NONBLOCK`	Open in nonblocking mode

One of the purposes of the *oflag* argument is to determine whether we are opening an existing queue or creating and opening a new queue. If *oflag* doesn't include `O_CREAT`, we are opening an existing queue. If *oflag* includes `O_CREAT`, a new, empty queue is created if one with the given *name* doesn't already exist. If *oflag* specifies both `O_CREAT` and `O_EXCL`, and a queue with the given *name* already exists, then *mq_open()* fails.

The *oflag* argument also indicates the kind of access that the calling process will make to the message queue, by specifying exactly one of the values `O_RDONLY`, `O_WRONLY`, or `O_RDWR`.

The remaining flag value, `O_NONBLOCK`, causes the queue to be opened in nonblocking mode. If a subsequent call to *mq_receive()* or *mq_send()* can't be performed without blocking, the call will fail immediately with the error `EAGAIN`.

If *mq_open()* is being used to open an existing message queue, the call requires only two arguments. However, if `O_CREAT` is specified in *flags*, two further arguments are required: *mode* and *attr*. (If the queue specified by *name* already exists, these two arguments are ignored.) These arguments are used as follows:

- The *mode* argument is a bit mask that specifies the permissions to be placed on the new message queue. The bit values that may be specified are the same as for files (Table 15-4, on page 295), and, as with *open()*, the value in *mode* is masked against the process umask (Section 15.4.6). To read from a queue (*mq_receive()*), read permission must be granted to the corresponding class of user; to write to a queue (*mq_send()*), write permission is required.
- The *attr* argument is an *mq_attr* structure that specifies attributes for the new message queue. If *attr* is `NULL`, the queue is created with implementation-defined default attributes. We describe the *mq_attr* structure in Section 52.4.

Upon successful completion, *mq_open()* returns a *message queue descriptor*, a value of type *mqd_t*, which is used in subsequent calls to refer to this open message queue. The only stipulation that SUSv3 makes about this data type is that it may not be an array; that is, it is guaranteed to be a type that can be used in an assignment statement or passed by value as a function argument. (On Linux, *mqd_t* is an *int*, but, for example, on Solaris it is defined as *void **.)

An example of the use of *mq_open()* is provided in Listing 52-2.

Effect of *fork()*, *exec()*, and process termination on message queue descriptors

During a *fork()*, the child process receives copies of its parent's message queue descriptors, and these descriptors refer to the same open message queue descriptions.

(We explain message queue descriptions in Section 52.3.) The child doesn't inherit any of its parent's message notification registrations.

When a process performs an *exec()* or terminates, all of its open message queue descriptors are closed. As a consequence of closing its message queue descriptors, all of the process's message notification registrations on the corresponding queues are deregistered.

Closing a message queue

The *mq_close()* function closes the message queue descriptor *mqdes*.

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

Returns 0 on success, or –1 on error

If the calling process has registered via *mqdes* for message notification from the queue (Section 52.6), then the notification registration is automatically removed, and another process can subsequently register for message notification from the queue.

A message queue descriptor is automatically closed when a process terminates or calls *exec()*. As with file descriptors, we should explicitly close message queue descriptors that are no longer required, in order to prevent the process from running out of message queue descriptors.

As *close()* for files, closing a message queue doesn't delete it. For that purpose, we need *mq_unlink()*, which is the message queue analog of *unlink()*.

Removing a message queue

The *mq_unlink()* function removes the message queue identified by *name*, and marks the queue to be destroyed once all processes cease using it (this may mean immediately, if all processes that had the queue open have already closed it).

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

Returns 0 on success, or –1 on error

Listing 52-1 demonstrates the use of *mq_unlink()*.

Listing 52-1: Using *mq_unlink()* to unlink a POSIX message queue

pmsg/pmsg_unlink.c

```
#include <mqueue.h>
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
```

```
    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s mq-name\n", argv[0]);

    if (mq_unlink(argv[1]) == -1)
        errExit("mq_unlink");
    exit(EXIT_SUCCESS);
}
```

pmsg/pmsg_unlink.c

52.3 Relationship Between Descriptors and Message Queues

The relationship between a message queue descriptor and an open message queue is analogous to the relationship between a file descriptor and an open file (Figure 5-2, on page 95). A message queue descriptor is a per-process handle that refers to an entry in the system-wide table of open message queue descriptions, and this entry in turn refers to a message queue object. This relationship is illustrated in Figure 52-1.

> On Linux, POSIX message queues are implemented as i-nodes in a virtual file system, and message queue descriptors and open message queue descriptions are implemented as file descriptors and open file descriptions, respectively. However, these are implementation details that are not required by SUSv3 and don't hold true on some other UNIX implementations. Nevertheless, we return to this point in Section 52.7, because Linux provides some nonstandard features that are made possible by this implementation.

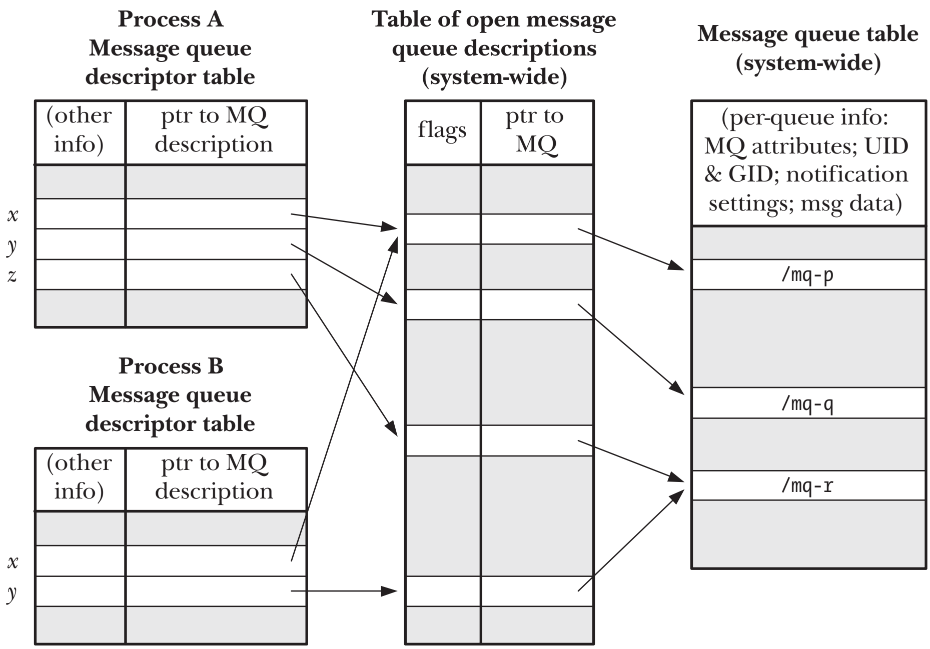

Figure 52-1: Relationship between kernel data structures for POSIX message queues

Figure 52-1 helps clarify a number of details of the use of message queue descriptors (all of which are analogous to the use to file descriptors):

- An open message queue description has an associated set of flags. SUSv3 specifies only one such flag, `O_NONBLOCK`, which determines whether I/O is nonblocking.
- Two processes can hold message queue descriptors (descriptor *x* in the diagram) that refer to the same open message queue description. This can occur because a process opens a message queue and then calls *fork()*. These descriptors share the state of the `O_NONBLOCK` flag.
- Two processes can hold open message queue descriptors that refer to different message queue descriptions that refer to the same message queue (e.g., descriptor *z* in process A and descriptor *y* in process B both refer to `/mq-r`). This occurs because the two processes each used *mq_open()* to open the same queue.

52.4 Message Queue Attributes

The *mq_open()*, *mq_getattr()*, and *mq_setattr()* functions all permit an argument that is a pointer to an *mq_attr* structure. This structure is defined in `<mqueue.h>` as follows:

```
struct mq_attr {
    long mq_flags;        /* Message queue description flags: 0 or
                             O_NONBLOCK [mq_getattr(), mq_setattr()] */
    long mq_maxmsg;       /* Maximum number of messages on queue
                             [mq_open(), mq_getattr()] */
    long mq_msgsize;      /* Maximum message size (in bytes)
                             [mq_open(), mq_getattr()] */
    long mq_curmsgs;      /* Number of messages currently in queue
                             [mq_getattr()] */
};
```

Before we look at the *mq_attr* structure in detail, it is worth noting the following points:

- Only some of the fields are used by each of the three functions. The fields used by each function are indicated in the comments accompanying the structure definition above.
- The structure contains information about the open message queue description (*mq_flags*) associated with a message descriptor and information about the queue referred to by that descriptor (*mq_maxmsg*, *mq_msgsize*, *mq_curmsgs*).
- Some of the fields contain information that is fixed at the time the queue is created with *mq_open()* (*mq_maxmsg* and *mq_msgsize*); the others return information about the current state of the message queue description (*mq_flags*) or message queue (*mq_curmsgs*).

Setting message queue attributes during queue creation

When we create a message queue with *mq_open()*, the following *mq_attr* fields determine the attributes of the queue:

- The *mq_maxmsg* field defines the limit on the number of messages that can be placed on the queue using *mq_send()*. This value must be greater than 0.

- The *mq_msgsize* field defines the upper limit on the size of each message that may be placed on the queue. This value must be greater than 0.

Together, these two values allow the kernel to determine the maximum amount of memory that this message queue may require.

The *mq_maxmsg* and *mq_msgsize* attributes are fixed when a message queue is created; they can't subsequently be changed. In Section 52.8, we describe two `/proc` files that place system-wide limits on the values that can be specified for the *mq_maxmsg* and *mq_msgsize* attributes.

The program in Listing 52-2 provides a command-line interface to the *mq_open()* function and shows how the *mq_attr* structure is used with *mq_open()*.

Two command-line options allow message queue attributes to be specified: *–m* for *mq_maxmsg* and *–s* for *mq_msgsize*. If either of these options is supplied, a non-`NULL` *attrp* argument is passed to *mq_open()*. Some default values are assigned to the fields of the *mq_attr* structure to which *attrp* points, in case only one of the *–m* and *–s* options is specified on the command line. If neither of these options is supplied, *attrp* is specified as `NULL` when calling *mq_open()*, which causes the queue to be created with the implementation-defined defaults for the queue attributes.

Listing 52-2: Creating a POSIX message queue

pmsg/pmsg_create.c

```
#include <mqueue.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "tlpi_hdr.h"

static void
usageError(const char *progName)
{
    fprintf(stderr, "Usage: %s [-cx] [-m maxmsg] [-s msgsize] mq-name "
            "[octal-perms]\n", progName);
    fprintf(stderr, "    -c          Create queue (O_CREAT)\n");
    fprintf(stderr, "    -m maxmsg   Set maximum # of messages\n");
    fprintf(stderr, "    -s msgsize  Set maximum message size\n");
    fprintf(stderr, "    -x          Create exclusively (O_EXCL)\n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
    int flags, opt;
    mode_t perms;
    mqd_t mqd;
    struct mq_attr attr, *attrp;

    attrp = NULL;
    attr.mq_maxmsg = 50;
    attr.mq_msgsize = 2048;
    flags = O_RDWR;
```

```
    /* Parse command-line options */

    while ((opt = getopt(argc, argv, "cm:s:x")) != -1) {
        switch (opt) {
        case 'c':
            flags |= O_CREAT;
            break;

        case 'm':
            attr.mq_maxmsg = atoi(optarg);
            attrp = &attr;
            break;

        case 's':
            attr.mq_msgsize = atoi(optarg);
            attrp = &attr;
            break;

        case 'x':
            flags |= O_EXCL;
            break;

        default:
            usageError(argv[0]);
        }
    }

    if (optind >= argc)
        usageError(argv[0]);

    perms = (argc <= optind + 1) ? (S_IRUSR | S_IWUSR) :
                getInt(argv[optind + 1], GN_BASE_8, "octal-perms");

    mqd = mq_open(argv[optind], flags, perms, attrp);
    if (mqd == (mqd_t) -1)
        errExit("mq_open");

    exit(EXIT_SUCCESS);
}
```

pmsg/pmsg_create.c

Retrieving message queue attributes

The *mq_getattr()* function returns an *mq_attr* structure containing information about the message queue description and the message queue associated with the descriptor *mqdes*.

```
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *attr);
```

Returns 0 on success, or –1 on error

In addition to the *mq_maxmsg* and *mq_msgsize* fields, which we have already described, the following fields are returned in the structure pointed to by *attr*:

mq_flags
: These are flags for the open message queue description associated with the descriptor *mqdes*. Only one such flag is specified: `O_NONBLOCK`. This flag is initialized from the *oflag* argument of *mq_open()*, and can be changed using *mq_setattr()*.

mq_curmsgs
: This is the number of messages that are currently in the queue. This information may already have changed by the time *mq_getattr()* returns, if other processes are reading messages from the queue or writing messages to it.

The program in Listing 52-3 employs *mq_getattr()* to retrieve the attributes for the message queue specified in its command-line argument, and then displays those attributes on standard output.

Listing 52-3: Retrieving POSIX message queue attributes

pmsg/pmsg_getattr.c

```
#include <mqueue.h>
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    mqd_t mqd;
    struct mq_attr attr;

    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s mq-name\n", argv[0]);

    mqd = mq_open(argv[1], O_RDONLY);
    if (mqd == (mqd_t) -1)
        errExit("mq_open");

    if (mq_getattr(mqd, &attr) == -1)
        errExit("mq_getattr");

    printf("Maximum # of messages on queue:   %ld\n", attr.mq_maxmsg);
    printf("Maximum message size:             %ld\n", attr.mq_msgsize);
    printf("# of messages currently on queue: %ld\n", attr.mq_curmsgs);
    exit(EXIT_SUCCESS);
}
```

pmsg/pmsg_getattr.c

In the following shell session, we use the program in Listing 52-2 to create a message queue with implementation-defined default attributes (i.e., the final argument to *mq_open()* is `NULL`), and then use the program in Listing 52-3 to display the queue attributes so that we can see the default settings on Linux.

```
$ ./pmsg_create -cx /mq
$ ./pmsg_getattr /mq
Maximum # of messages on queue:   10
Maximum message size:             8192
# of messages currently on queue: 0
$ ./pmsg_unlink /mq                                     Remove message queue
```

From the above output, we see that the Linux default values for *mq_maxmsg* and *mq_msgsize* are 10 and 8192, respectively.

There is a wide variation in the implementation-defined defaults for *mq_maxmsg* and *mq_msgsize*. Portable applications generally need to choose explicit values for these attributes, rather than relying on the defaults.

Modifying message queue attributes

The *mq_setattr()* function sets attributes of the message queue description associated with the message queue descriptor *mqdes*, and optionally returns information about the message queue.

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *newattr,
               struct mq_attr *oldattr);
```

Returns 0 on success, or –1 on error

The *mq_setattr()* function performs the following tasks:

- It uses the *mq_flags* field in the *mq_attr* structure pointed to by *newattr* to change the flags of the message queue description associated with the descriptor *mqdes*.
- If *oldattr* is non-NULL, it returns an *mq_attr* structure containing the previous message queue description flags and message queue attributes (i.e., the same task as is performed by *mq_getattr()*).

The only attribute that SUSv3 specifies that can be changed using *mq_setattr()* is the state of the O_NONBLOCK flag.

Allowing for the possibility that a particular implementation may define other modifiable flags, or that SUSv3 may add new flags in the future, a portable application should change the state of the O_NONBLOCK flag by using *mq_getattr()* to retrieve the *mq_flags* value, modifying the O_NONBLOCK bit, and calling *mq_setattr()* to change the *mq_flags* settings. For example, to enable O_NONBLOCK, we would do the following:

```
if (mq_getattr(mqd, &attr) == -1)
    errExit("mq_getattr");
attr.mq_flags |= O_NONBLOCK;
if (mq_setattr(mqd, &attr, NULL) == -1)
    errExit("mq_getattr");
```

52.5 Exchanging Messages

In this section, we look at the functions that are used to send messages to and receive messages from a queue.

52.5.1 Sending Messages

The *mq_send()* function adds the message in the buffer pointed to by *msg_ptr* to the message queue referred to by the descriptor *mqdes*.

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
            unsigned int msg_prio);
```

Returns 0 on success, or –1 on error

The *msg_len* argument specifies the length of the message pointed to by *msg_ptr*. This value must be less than or equal to the *mq_msgsize* attribute of the queue; otherwise, *mq_send()* fails with the error `EMSGSIZE`. Zero-length messages are permitted.

Each message has a nonnegative integer priority, specified by the *msg_prio* argument. Messages are ordered within the queue in descending order of priority (i.e., 0 is the lowest priority). When a new message is added to the queue, it is placed after any other messages of the same priority. If an application doesn't need to use message priorities, it is sufficient to always specify *msg_prio* as 0.

> As noted at the beginning of this chapter, the type attribute of System V messages provides different functionality. System V messages are always queued in FIFO order, but *msgrcv()* allows us to select messages in various ways: in FIFO order, by exact type, or by highest type less than or equal to some value.

SUSv3 allows an implementation to advertise its upper limit for message priorities, either by defining the constant `MQ_PRIO_MAX` or via the return from *sysconf(_SC_MQ_PRIO_MAX)*. SUSv3 requires this limit to be at least 32 (`_POSIX_MQ_PRIO_MAX`); that is, priorities at least in the range 0 to 31 are available. However, the actual range on implementations is highly variable. For example, on Linux, this constant has the value 32,768; on Solaris, it is 32; and on Tru64, it is 256.

If the message queue is already full (i.e., the *mq_maxmsg* limit for the queue has been reached), then a further *mq_send()* either blocks until space becomes available in the queue, or, if the `O_NONBLOCK` flag is in effect, fails immediately with the error `EAGAIN`.

The program in Listing 52-4 provides a command-line interface to the *mq_send()* function. We demonstrate the use of this program in the next section.

Listing 52-4: Writing a message to a POSIX message queue

pmsg/pmsg_send.c

```
#include <mqueue.h>
#include <fcntl.h>              /* For definition of O_NONBLOCK */
#include "tlpi_hdr.h"

static void
usageError(const char *progName)
{
    fprintf(stderr, "Usage: %s [-n] name msg [prio]\n", progName);
    fprintf(stderr, "    -n          Use O_NONBLOCK flag\n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
    int flags, opt;
    mqd_t mqd;
    unsigned int prio;

    flags = O_WRONLY;
    while ((opt = getopt(argc, argv, "n")) != -1) {
        switch (opt) {
        case 'n':   flags |= O_NONBLOCK;        break;
        default:    usageError(argv[0]);
        }
    }

    if (optind + 1 >= argc)
        usageError(argv[0]);

    mqd = mq_open(argv[optind], flags);
    if (mqd == (mqd_t) -1)
        errExit("mq_open");

    prio = (argc > optind + 2) ? atoi(argv[optind + 2]) : 0;

    if (mq_send(mqd, argv[optind + 1], strlen(argv[optind + 1]), prio) == -1)
        errExit("mq_send");
    exit(EXIT_SUCCESS);
}
```

pmsg/pmsg_send.c

52.5.2 Receiving Messages

The *mq_receive()* function removes the oldest message with the highest priority from the message queue referred to by *mqdes* and returns that message in the buffer pointed to by *msg_ptr*.

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
                   unsigned int *msg_prio);
```

Returns number of bytes in received message on success, or –1 on error

The *msg_len* argument is used by the caller to specify the number of bytes of space available in the buffer pointed to by *msg_ptr*.

Regardless of the actual size of the message, *msg_len* (and thus the size of the buffer pointed to by *msg_ptr*) must be greater than or equal to the *mq_msgsize* attribute of the queue; otherwise, *mq_receive()* fails with the error EMSGSIZE. If we don't know the value of the *mq_msgsize* attribute of a queue, we can obtain it using *mq_getattr()*. (In an application consisting of cooperating processes, the use of *mq_getattr()* can usually be dispensed with, because the application can typically decide on a queue's *mq_msgsize* setting in advance.)

If *msg_prio* is not NULL, then the priority of the received message is copied into the location pointed to by *msg_prio*.

If the message queue is currently empty, then *mq_receive()* either blocks until a message becomes available, or, if the O_NONBLOCK flag is in effect, fails immediately with the error EAGAIN. (There is no equivalent of the pipe behavior where a reader sees end-of-file if there are no writers.)

The program in Listing 52-5 provides a command-line interface to the *mq_receive()* function. The command format for this program is shown in the *usageError()* function.

The following shell session demonstrates the use of the programs in Listing 52-4 and Listing 52-5. We begin by creating a message queue and sending a few messages with different priorities:

```
$ ./pmsg_create -cx /mq
$ ./pmsg_send /mq msg-a 5
$ ./pmsg_send /mq msg-b 0
$ ./pmsg_send /mq msg-c 10
```

We then execute a series of commands to retrieve messages from the queue:

```
$ ./pmsg_receive /mq
Read 5 bytes; priority = 10
msg-c
$ ./pmsg_receive /mq
Read 5 bytes; priority = 5
msg-a
$ ./pmsg_receive /mq
Read 5 bytes; priority = 0
msg-b
```

As we can see from the above output, the messages were retrieved in order of priority.

At this point, the queue is now empty. When we perform another blocking receive, the operation blocks:

```
$ ./pmsg_receive /mq
Blocks; we type Control-C to terminate the program
```

On the other hand, if we perform a nonblocking receive, the call returns immediately with a failure status:

```
$ ./pmsg_receive -n /mq
ERROR [EAGAIN/EWOULDBLOCK Resource temporarily unavailable] mq_receive
```

Listing 52-5: Reading a message from a POSIX message queue

pmsg/pmsg_receive.c

```
#include <mqueue.h>
#include <fcntl.h>              /* For definition of O_NONBLOCK */
#include "tlpi_hdr.h"

static void
usageError(const char *progName)
{
    fprintf(stderr, "Usage: %s [-n] name\n", progName);
    fprintf(stderr, "    -n           Use O_NONBLOCK flag\n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char *argv[])
{
    int flags, opt;
    mqd_t mqd;
    unsigned int prio;
    void *buffer;
    struct mq_attr attr;
    ssize_t numRead;

    flags = O_RDONLY;
    while ((opt = getopt(argc, argv, "n")) != -1) {
        switch (opt) {
        case 'n':   flags |= O_NONBLOCK;        break;
        default:    usageError(argv[0]);
        }
    }

    if (optind >= argc)
        usageError(argv[0]);

    mqd = mq_open(argv[optind], flags);
    if (mqd == (mqd_t) -1)
        errExit("mq_open");

    if (mq_getattr(mqd, &attr) == -1)
        errExit("mq_getattr");

    buffer = malloc(attr.mq_msgsize);
    if (buffer == NULL)
        errExit("malloc");
```

```
    numRead = mq_receive(mqd, buffer, attr.mq_msgsize, &prio);
    if (numRead == -1)
        errExit("mq_receive");

    printf("Read %ld bytes; priority = %u\n", (long) numRead, prio);
    if (write(STDOUT_FILENO, buffer, numRead) == -1)
        errExit("write");
    write(STDOUT_FILENO, "\n", 1);

    exit(EXIT_SUCCESS);
}
```

pmsg/pmsg_receive.c

52.5.3 Sending and Receiving Messages with a Timeout

The *mq_timedsend()* and *mq_timedreceive()* functions are exactly like *mq_send()* and *mq_receive()*, except that if the operation can't be performed immediately, and the O_NONBLOCK flag is not in effect for the message queue description, then the *abs_timeout* argument specifies a limit on the time for which the call will block.

```
#define _XOPEN_SOURCE 600
#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
                 unsigned int msg_prio, const struct timespec *abs_timeout);
```

Returns 0 on success, or –1 on error

```
ssize_t mq_timedreceive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
                 unsigned int *msg_prio, const struct timespec *abs_timeout);
```

Returns number of bytes in received message on success, or –1 on error

The *abs_timeout* argument is a *timespec* structure (Section 23.4.2) that specifies the timeout as an absolute value in seconds and nanoseconds since the Epoch. To perform a relative timeout, we can fetch the current value of the CLOCK_REALTIME clock using *clock_gettime()* and add the required amount to that value to produce a suitably initialized *timespec* structure.

If a call to *mq_timedsend()* or *mq_timedreceive()* times out without being able to complete its operation, then the call fails with the error ETIMEDOUT.

On Linux, specifying *abs_timeout* as NULL means an infinite timeout. However, this behavior is not specified in SUSv3, and portable applications can't rely on it.

The *mq_timedsend()* and *mq_timedreceive()* functions originally derive from POSIX.1d (1999) and are not available on all UNIX implementations.

52.6 Message Notification

A feature that distinguishes POSIX message queues from their System V counterparts is the ability to receive asynchronous notification of the availability of a message on a previously empty queue (i.e., when the queue transitions from being empty to nonempty). This feature means that instead of making a blocking *mq_receive()* call

BIBLIOGRAPHY

Aho, A.V., Kernighan, B.W., and Weinberger, P. J. 1988. *The AWK Programming Language*. Addison-Wesley, Reading, Massachusetts.

Albitz, P., and Liu, C. 2006. *DNS and BIND (5th edition)*. O'Reilly, Sebastopol, California.

Anley, C., Heasman, J., Lindner, F., and Richarte, G. 2007. *The Shellcoder's Handbook: Discovering and Exploiting Security Holes*. Wiley, Indianapolis, Indiana.

Bach, M. 1986. *The Design of the UNIX Operating System*. Prentice Hall, Englewood Cliffs, New Jersey.

Bhattiprolu, S., Biederman, E.W., Hallyn, S., and Lezcano, D. 2008. "Virtual Servers and Checkpoint/Restart in Mainstream Linux," *ACM SIGOPS Operating Systems Review*, Vol. 42, Issue 5, July 2008, pages 104–113.

http://www.mnis.fr/fr/services/virtualisation/pdf/cr.pdf

Bishop, M. 2003. *Computer Security: Art and Science*. Addison-Wesley, Reading, Massachusetts.

Bishop, M. 2005. *Introduction to Computer Security*. Addison-Wesley, Reading, Massachusetts.

Borisov, N., Johnson, R., Sastry, N., and Wagner, D. 2005. "Fixing Races for Fun and Profit: How to abuse atime," *Proceedings of the 14th USENIX Security Symposium*.

http://www.cs.berkeley.edu/~nks/papers/races-usenix05.pdf

Bovet, D.P., and Cesati, M. 2005. *Understanding the Linux Kernel (3rd edition)*. O'Reilly, Sebastopol, California.

Butenhof, D.R. 1996. *Programming with POSIX Threads*. Addison-Wesley, Reading, Massachusetts.

Further information, as well as source code for the programs in this book, can be found at *http://homepage.mac.com/dbutenhof/Threads/Threads.html*.

Chen, H., Wagner, D., and Dean, D. 2002. "Setuid Demystified," *Proceedings of the 11th USENIX Security Symposium*.

http://www.cs.berkeley.edu/~daw/papers/setuid-usenix02.pdf

Comer, D.E. 2000. *Internetworking with TCP/IP Vol. I: Principles, Protocols, and Architecture (4th edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Further information about the *Internetworking with TCP/IP* book series (including source code) can be found at *http://www.cs.purdue.edu/homes/dec/netbooks.html*.

Comer, D.E., and Stevens, D.L. 1999. *Internetworking with TCP/IP Vol. II: Design, Implementation, and Internals (3rd edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Comer, D.E., and Stevens, D.L. 2000. *Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications, Linux/Posix Sockets Version*. Prentice Hall, Englewood Cliffs, New Jersey.

Corbet, J. 2002. "The Orlov block allocator." *Linux Weekly News*, 5 November 2002.

http://lwn.net/Articles/14633/

Corbet, J., Rubini, A., and Kroah-Hartman, G. 2005. *Linux Device Drivers (3rd edition)*. O'Reilly, Sebastopol, California.

http://lwn.net/Kernel/LDD3/

Crosby, S.A., and Wallach, D. S. 2003. "Denial of Service via Algorithmic Complexity Attacks," *Proceedings of the 12th USENIX Security Symposium*.

http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003.pdf

Deitel, H.M., Deitel, P. J., and Choffnes, D. R. 2004. *Operating Systems (3rd edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Dijkstra, E.W. 1968. "Cooperating Sequential Processes," *Programming Languages*, ed. F. Genuys, Academic Press, New York.

Drepper, U. 2004 (a). "Futexes Are Tricky."

http://people.redhat.com/drepper/futex.pdf

Drepper, U. 2004 (b). "How to Write Shared Libraries."

http://people.redhat.com/drepper/dsohowto.pdf

Drepper, U. 2007. "What Every Programmer Should Know About Memory."

http://people.redhat.com/drepper/cpumemory.pdf

Drepper, U. 2009. "Defensive Programming for Red Hat Enterprise Linux."

http://people.redhat.com/drepper/defprogramming.pdf

Erickson, J.M. 2008. *Hacking: The Art of Exploitation (2nd edition)*. No Starch Press, San Francisco, California.

Floyd, S. 1994. "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, Vol. 24, No. 5, October 1994, pages 10–23.

http://www.icir.org/floyd/papers/tcp_ecn.4.pdf

Franke, H., Russell, R., and Kirkwood, M. 2002. "Fuss, Futexes and Furwocks: Fast Userlevel Locking in Linux," *Proceedings of the Ottawa Linux Symposium 2002*.

http://www.kernel.org/doc/ols/2002/ols2002-pages-479-495.pdf

Frisch, A. 2002. *Essential System Administration (3rd edition)*. O'Reilly, Sebastopol, California.

Gallmeister, B.O. 1995. *POSIX.4: Programming for the Real World*. O'Reilly, Sebastopol, California.

Gammo, L., Brecht, T., Shukla, A., and Pariag, D. 2004. "Comparing and Evaluating epoll, select, and poll Event Mechanisms," *Proceedings of the Ottawa Linux Symposium 2002*.

http://www.kernel.org/doc/ols/2004/ols2004v1-pages-215-226.pdf

Gancarz, M. 2003. *Linux and the Unix Philosophy*. Digital Press.

Garfinkel, S., Spafford, G., and Schwartz, A. 2003. *Practical Unix and Internet Security (3rd edition)*. O'Reilly, Sebastopol, California.

Gont, F. 2008. *Security Assessment of the Internet Protocol*. UK Centre for the Protection of the National Infrastructure.

http://www.cpni.gov.uk/Docs/InternetProtocol.pdf

Gont, F. 2009 (a). *Security Assessment of the Transmission Control Protocol (TCP)*. CPNI Technical Note 3/2009. UK Centre for the Protection of the National Infrastructure.

http://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf

Gont, F., and Yourtchenko, A. 2009 (b). "On the implementation of TCP urgent data." Internet draft, 20 May 2009.

http://www.gont.com.ar/drafts/urgent-data/

Goodheart, B., and Cox, J. 1994. *The Magic Garden Explained: The Internals of UNIX SVR4*. Prentice Hall, Englewood Cliffs, New Jersey.

Goralski, W. 2009. *The Illustrated Network: How TCP/IP Works in a Modern Network*. Morgan Kaufmann, Burlington, Massachusetts.

Gorman, M. 2004. *Understanding the Linux Virtual Memory Manager*. Prentice Hall, Upper Saddle River, New Jersey.

Available online at *http://www.phptr.com/perens*.

Grünbacher, A. 2003. "POSIX Access Control Lists on Linux," *Proceedings of USENIX 2003/Freenix Track*, pages 259–272.

http://www.suse.de/~agruen/acl/linux-acls/online/

Gutmann, P. 1996. "Secure Deletion of Data from Magnetic and Solid-State Memory," *Proceedings of the 6th USENIX Security Symposium*.

http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html

Hallyn, S. 2007. "POSIX file capabilities: Parceling the power of root."

http://www.ibm.com/developerworks/library/l-posixcap.html

Harbison, S., and Steele, G. 2002. *C: A Reference Manual (5th edition)*. Prentice Hall, Englewood Cliffs, New Jersey.

Herbert, T.F. 2004. *The Linux TCP/IP Stack: Networking for Embedded Systems*. Charles River Media, Hingham, Massachusetts.

Hubička, J. 2003. "Porting GCC to the AMD64 Architecture," *Proceedings of the First Annual GCC Developers' Summit*.

http://www.ucw.cz/~hubicka/papers/amd64/index.html

Johnson, M.K., and Troan, E.W. 2005. *Linux Application Development (2nd edition)*. Addison-Wesley, Reading, Massachusetts.

Josey, A. (ed.). 2004. *The Single UNIX Specification, Authorized Guide to Version 3*. The Open Group.

See *http://www.unix-systems.org/version3/theguide.html* for details on ordering this guide. A newer version of this guide (published in 2010) for version 4 of the specification can be ordered online at *http://www.opengroup.org/bookstore/catalog/*.

Kent, A., and Mogul, J.C. 1987. "Fragmentation Considered Harmful," *ACM Computer Communication Review*, Vol. 17, No. 5, August 1987.

http://www.acm.org/sigcomm/ccr/archive/1995/jan95/ccr-9501-mogulf1.pdf

Kernighan, B.W., and Ritchie, D.M. 1988. *The C Programming Language (2nd edition)*. Prentice Hall, Englewood Cliffs, New Jersey.

Kopparapu, C. 2002. *Load Balancing Servers, Firewalls, and Caches*. John Wiley and Sons.

Kozierok, C.M. 2005. *The TCP/IP Guide*. No Starch Press, San Francisco, California.

http://www.tcpipguide.com/

Kroah-Hartman, G. 2003. "udev—A Userspace Implementation of devfs," *Proceedings of the 2003 Linux Symposium*.

http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf

Kumar, A., Cao, M., Santos, J., and Dilger, A. 2008. "Ext4 block and inode allocator improvements," *Proceedings of the 2008 Linux Symposium*, Ottawa, Canada.

http://ols.fedoraproject.org/OLS/Reprints-2008/kumar-reprint.pdf

Lemon, J. 2001. "Kqueue: A generic and scalable event notification facility," *Proceedings of USENIX 2001/Freenix Track*.

http://people.freebsd.org/~jlemon/papers/kqueue_freenix.pdf

Lemon, J. 2002. "Resisting SYN flood DoS attacks with a SYN cache," *Proceedings of USENIX BSDCon 2002*.

http://people.freebsd.org/~jlemon/papers/syncache.pdf

Levine, J. 2000. *Linkers and Loaders*. Morgan Kaufmann, San Francisco, California.

http://www.iecc.com/linker/

Lewine, D. 1991. *POSIX Programmer's Guide*. O'Reilly, Sebastopol, California.

Liang, S. 1999. *The Java Native Interface: Programmer's Guide and Specification*. Addison-Wesley, Reading, Massachusetts.

http://java.sun.com/docs/books/jni/

Libes, D., and Ressler, S. 1989. *Life with UNIX: A Guide for Everyone*. Prentice Hall, Englewood Cliffs, New Jersey.

Lions, J. 1996. *Lions' Commentary on UNIX 6th Edition with Source Code*. Peer-to-Peer Communications, San Jose, California.

[Lions, 1996] was originally produced by an Australian academic, the late John Lions, in 1977 for use in an operating systems class that he taught. At that time, it could not be formally published because of licensing restrictions. Nevertheless, pirated photocopies became widely distributed within the UNIX community, and, in Dennis Ritchie's words, "educated a generation" of UNIX programmers.

Love, R. 2010. *Linux Kernel Development (3rd edition)*. Addison-Wesley, Reading, Massachusetts.

Lu, H.J. 1995. "ELF: From the Programmer's Perspective."

http://www.trunix.org/programlama/os/elf-hl/Documentation/elf/elf.html

Mann, S., and Mitchell, E.L. 2003. *Linux System Security (2nd edition)*. Prentice Hall, Englewood Cliffs, New Jersey.

Matloff, N. and Salzman, P.J. 2008. *The Art of Debugging with GDB, DDD, and Eclipse*. No Starch Press, San Francisco, California.

Maxwell, S. 1999. *Linux Core Kernel Commentary*. Coriolis, Scottsdale, Arizona.

This book provides an annotated listing of selected parts of the Linux 2.2.5 kernel.

McKusick, M.K., Joy, W.N., Leffler, S.J., and Fabry, R.S. 1984. "A fast file system for UNIX," *ACM Transactions on Computer Systems*, Vol. 2, Issue 3 (August).

This paper can be found online at a variety of locations.

McKusick, M.K. 1999. "Twenty years of Berkeley Unix," *Open Sources: Voices from the Open Source Revolution*, C. DiBona, S. Ockman, and M. Stone (eds.). O'Reilly, Sebastopol, California.

McKusick, M.K., Bostic, K., and Karels, M.J. 1996. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley, Reading, Massachusetts.

McKusick, M.K., and Neville-Neil, G.V. 2005. *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley, Reading, Massachusetts.

Mecklenburg, R. 2005. *Managing Projects with GNU Make (3rd edition)*. O'Reilly, Sebastopol, California.

Mills, D.L. 1992. "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, March 1992.

http://www.rfc-editor.org/rfc/rfc1305.txt

Mochel, P. "The sysfs Filesystem," *Proceedings of the Ottawa Linux Symposium 2002*.

Mosberger, D., and Eranian, S. 2002. *IA-64 Linux Kernel: Design and Implementation*. Prentice Hall, Upper Saddle River, New Jersey.

Peek, J., Todino-Gonguet, G., and Strang, J. 2001. *Learning the UNIX Operating System (5th edition)*. O'Reilly, Sebastopol, California.

Peikari, C., and Chuvakin, A. 2004. *Security Warrior*. O'Reilly, Sebastopol, California.

Plauger, P.J. 1992. *The Standard C Library*. Prentice Hall, Englewood Cliffs, New Jersey.

Quarterman, J.S., and Wilhelm, S. 1993. *UNIX, Posix, and Open Systems: The Open Standards Puzzle*. Addison-Wesley, Reading, Massachusetts.

Ritchie, D.M. 1984. "The Evolution of the UNIX Time-sharing System," *AT&T Bell Laboratories Technical Journal*, 63, No. 6 Part 2 (October 1984), pages 1577–93.

Dennis Ritchie's home page (*http://www.cs.bell-labs.com/who/dmr/index.html*) provides a pointer to an online version of this paper, as well as [Ritchie & Thompson, 1974], and has many other pieces of UNIX history.

Ritchie, D.M., and Thompson, K.L. 1974. “The Unix Time-Sharing System,” *Communications of the ACM*, 17 (July 1974), pages 365–375.

Robbins, K.A., and Robbins, S. 2003. *UNIX Systems Programming: Communication, Concurrency, and Threads (2nd edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Rochkind, M.J. 1985. *Advanced UNIX Programming*. Prentice Hall, Englewood Cliffs, New Jersey.

Rochkind, M.J. 2004. *Advanced UNIX Programming (2nd edition)*. Addison-Wesley, Reading, Massachusetts.

Rosen, L. 2005. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, Upper Saddle River, New Jersey.

St. Laurent, A.M. 2004. *Understanding Open Source and Free Software Licensing*. O’Reilly, Sebastopol, California.

Salus, P.H. 1994. *A Quarter Century of UNIX*. Addison-Wesley, Reading, Massachusetts.

Salus, P.H. 2008. *The Daemon, the Gnu, and the Penguin*. Addison-Wesley, Reading, Massachusetts.

http://www.groklaw.net/staticpages/index.php?page=20051013231901859
A short history of Linux, the BSDs, the HURD, and other free software projects.

Sarolahti, P., and Kuznetsov, A. 2002. “Congestion Control in Linux TCP,” *Proceedings of USENIX 2002/Freenix Track*.

http://www.cs.helsinki.fi/research/iwtcp/papers/linuxtcp.pdf

Schimmel, C. 1994. *UNIX Systems for Modern Architectures*. Addison-Wesley, Reading, Massachusetts.

Snader, J.C. 2000. *Effective TCP/IP Programming: 44 tips to improve your network programming*. Addison-Wesley, Reading, Massachusetts.

Stevens, W.R. 1992. *Advanced Programming in the UNIX Environment*. Addison-Wesley, Reading, Massachusetts.

Further information about all of the books of the late W. Richard Stevens (including program source code, with some modified code versions for Linux submitted by readers) can be found at *http://www.kohala.com/start/*.

Stevens, W.R. 1998. *UNIX Network Programming, Volume 1 (2nd edition): Networking APIs: Sockets and XTI*. Prentice Hall, Upper Saddle River, New Jersey.

Stevens, W.R. 1999. *UNIX Network Programming, Volume 2 (2nd edition): Interprocess Communications*. Prentice Hall, Upper Saddle River, New Jersey.

Stevens, W.R. 1994. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, Reading, Massachusetts.

Stevens, W.R. 1996. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*. Addison-Wesley, Reading, Massachusetts.

Stevens, W.R., Fenner, B., and Rudoff, A.M. 2004. *UNIX Network Programming, Volume 1 (3rd edition): The Sockets Networking API*. Addison-Wesley, Boston, Massachusetts.

Source code for this edition is available at *http://www.unpbook.com/*.

In most cases where we make reference to [Stevens et al., 2004] in this book, the same material can also found in [Stevens, 1998], the preceding edition of *UNIX Network Programming, Volume 1*.

Stevens, W.R., and Rago, S.A. 2005. *Advanced Programming in the UNIX Environment (2nd edition)*. Addison-Wesley, Boston, Massachusetts.

Stewart, R.R., and Xie, Q. 2001. *Stream Control Transmission Protocol (SCTP)*. Addison-Wesley, Reading, Massachusetts.

Stone, J., and Partridge, C. 2000. "When the CRC and the TCP Checksum Disagree," *Proceedings of SIGCOMM 2000*.

http://www.acm.org/sigcomm/sigcomm2000/conf/abstract/9-1.htm

Strang, J. 1986. *Programming with Curses*. O'Reilly, Sebastopol, California.

Strang, J., Mui, L., and O'Reilly, T. 1988. *Termcap & Terminfo (3rd edition)*. O'Reilly, Sebastopol, California.

Tanenbaum, A.S. 2007. *Modern Operating Systems (3rd edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Tanenbaum, A.S. 2002. *Computer Networks (4th edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Tanenbaum, A.S., and Woodhull, A.S. 2006. *Operating Systems: Design And Implementation (3rd edition)*. Prentice Hall, Upper Saddle River, New Jersey.

Torvalds, L.B., and Diamond, D. 2001. *Just for Fun: The Story of an Accidental Revolutionary*. HarperCollins, New York, New York.

Tsafrir, D., da Silva, D., and Wagner, D. "The Murky Issue of Changing Process Identity: Revising 'Setuid Demystified'," *;login: The USENIX Magazine*, June 2008.

http://www.usenix.org/publications/login/2008-06/pdfs/tsafrir.pdf

Vahalia, U. 1996. *UNIX Internals: The New Frontiers*. Prentice Hall, Upper Saddle River, New Jersey.

van der Linden, P. 1994. *Expert C Programming–Deep C Secrets*. Prentice Hall, Englewood Cliffs, New Jersey.

Vaughan, G.V., Elliston, B., Tromey, T., and Taylor, I.L. 2000. *GNU Autoconf, Automake, and Libtool*. New Riders, Indianapolis, Indiana.

http://sources.redhat.com/autobook/

Viega, J., and McGraw, G. 2002. *Building Secure Software*. Addison-Wesley, Reading, Massachusetts.

Viro, A. and Pai, R. 2006. "Shared-Subtree Concept, Implementation, and Applications in Linux," *Proceedings of the Ottawa Linux Symposium 2006*.

http://www.kernel.org/doc/ols/2006/ols2006v2-pages-209-222.pdf

Watson, R.N.M. 2000. "Introducing Supporting Infrastructure for Trusted Operating System Support in FreeBSD," *Proceedings of BSDCon 2000*.

http://www.trustedbsd.org/trustedbsd-bsdcon-2000.pdf

Williams, S. 2002. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly, Sebastopol, California.

Wright, G.R., and Stevens, W.R. 1995. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, Reading, Massachusetts.